

# Azure unleashed: ASP.NET Core <3 CQRS

**Andrea Saltarello**

Solution Architect @ Managed Designs S.r.l.

Microsoft Regional Director



@andysal74

.NET Present and Future

Grazie a



Partner



Communities



# Me.About();

- CEO @ [Managed Designs](#)
- Microsoft MVP since 2003
- Microsoft Regional Director
- Author (along with Dino) of [.NET: Architecting Applications for the Enterprise](#), by Microsoft Press
- Basically, a ~~software architect~~ developer eager to write code 😊

# Demos.About();

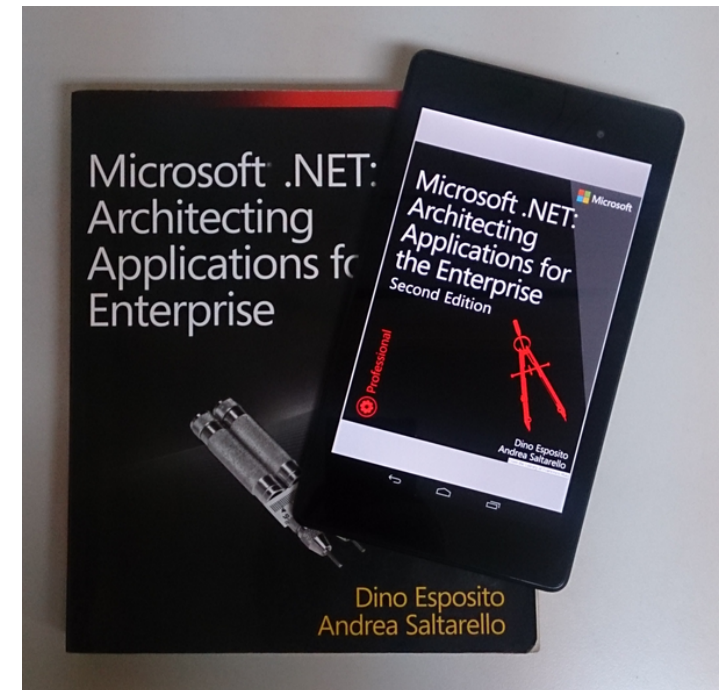
All demos (but one) are based on Merp, a GPL'ed Micro ERP I developed (and still do, to an extent) as the companion project for my book.

It's "free as in speech", so:

1. [Download](#) & unzip it
2. Open the .sln file in Visual Studio
3. Run Update-Database (x2)
4. Enjoy time travelling 😊

Newer builds will be published here:

<http://www.github.com/mastreeno>



# The (Relational) Lord of the Rings



A few fancy dressed blokes going on a jaunt





*It really became clear to me in the last couple of years that we need a new building block and that is the Domain Event.*

[Eric Evans]

*An event is something that has happened in the past.*

[Greg Young]

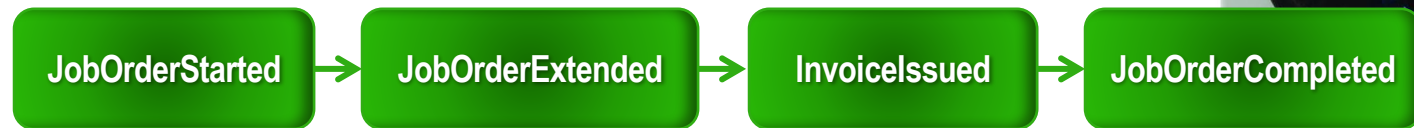
*A domain event ... captures the memory of something interesting which affects the domain*

[Martin Fowler]

# Event Sourcing in a nutshell

*Instead of focusing on a system's last known state, we might note down every occurring event: this way, we would be able to (re)build the state the system was in at any point in time just replaying those events*

To cut a long story short: we'd end up recording an **event stream**





# What's an event, anyway?

The (immutable) composition of:

- A (meaningful) name
- (Typed) Attributes

InvoiceIssued
DateOfIssue
Customer
Price

ProjectStarted
DateOfStart
ProjectId

ProjectRegistered
DateOfRegistration
DateOfEstimatedCompletion
ProjectId
CustomerId
Price

ProjectCompleted
DateOfCompletion
ProjectId

# DEMO

Event Stream

# Events vs. Relations

Although replaying a DBMS event log or adopting a temporal database would allow to restore a specific system state, we would miss the reason behind every occurred change nonetheless

INSERT INTO X (M, L, G) VALUES (1, 0, 1)

UPDATE X SET M=X, L=Y ... WHERE ...

UPDATE X SET M=42 ... WHERE ...

UPDATE X SET J=K ... WHERE ...

INSERT JobOrderStarted VALUES ()

INSERT JobOrderExtended VALUES ()

INSERT InvoiceIssued VALUES ()

INSERT JobOrderCompleted VALUES ()

# Event Stream vs. «My application»

*Still, my users are more interested in knowing a job order's balance or whether an invoice has been paid. (cit.)*

That is, we need a way to produce an entity state

# Event Sourcing <3 DDD

DDD's Aggregates provide a convenient way to encapsulate event management

***Aggregate:*** *A collection of objects that are bound together by a root entity, otherwise known as an aggregate root. The aggregate root guarantees the consistency of changes being made within the aggregate.*

[[Wikipedia](#)]

An aggregate is responsible for:

- encapsulating business logic pertaining to an “entity”
- generating events to have them available for saving
- replaying events in order to rebuild a specific state



# DEMO

Aggregates

# Aggregates vs. Events vs. Repos

```
var aggr = repository.GetByld<TAgr>(id);    //Triggers [time travelling] event replay
aggr.DoSomething();                        //Biz logic + events
repository.Save(aggr);                    //Updates the event stream
```

**Repository:** *Mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects. [DDD]*

# DEMO

Time Travelling

# Event Stream vs. «My application»

*Still<sup>2</sup>, my users are more interested in knowing a job order's balance or whether an invoice has been paid. Quickly.*

Ways to achieve that:

- Snapshots can help
- CQRS to the rescue: let's have a database storing the usual «last known system state» using it as a *read model*

# Enter CQRS

Acronym for **C**ommand **Q**uery **R**esponsibility **S**egregation

Basically, ad hoc application stacks for either writing or reading:

- “Command” stack writes events and snapshots
- “Read” stack reads from eventually consistent, reading purposes optimized database(s)



# CQRS: the “Read” side of the Force

*As a business unit manager, I want to collect credits due to unpaid outgoing invoices #ubiquitouslanguage #nuffsaid*

CQRS/ES wise, this user story could be implemented by means of the following real world C# code:

```
Database.OutgoingInvoices.  
    .PerBusinessUnit(businessUnitId)  
    .ExpiredOnly()  
    .Select(i => new {InvoiceNumber = i.Number, CustomerId = i.Customer.Id})  
    .AsParallel()  
    .ForAll(i => bus.Send(new CollectDebtCommand(i.InvoiceNumber, i.CustomerId)));
```

# DEMO

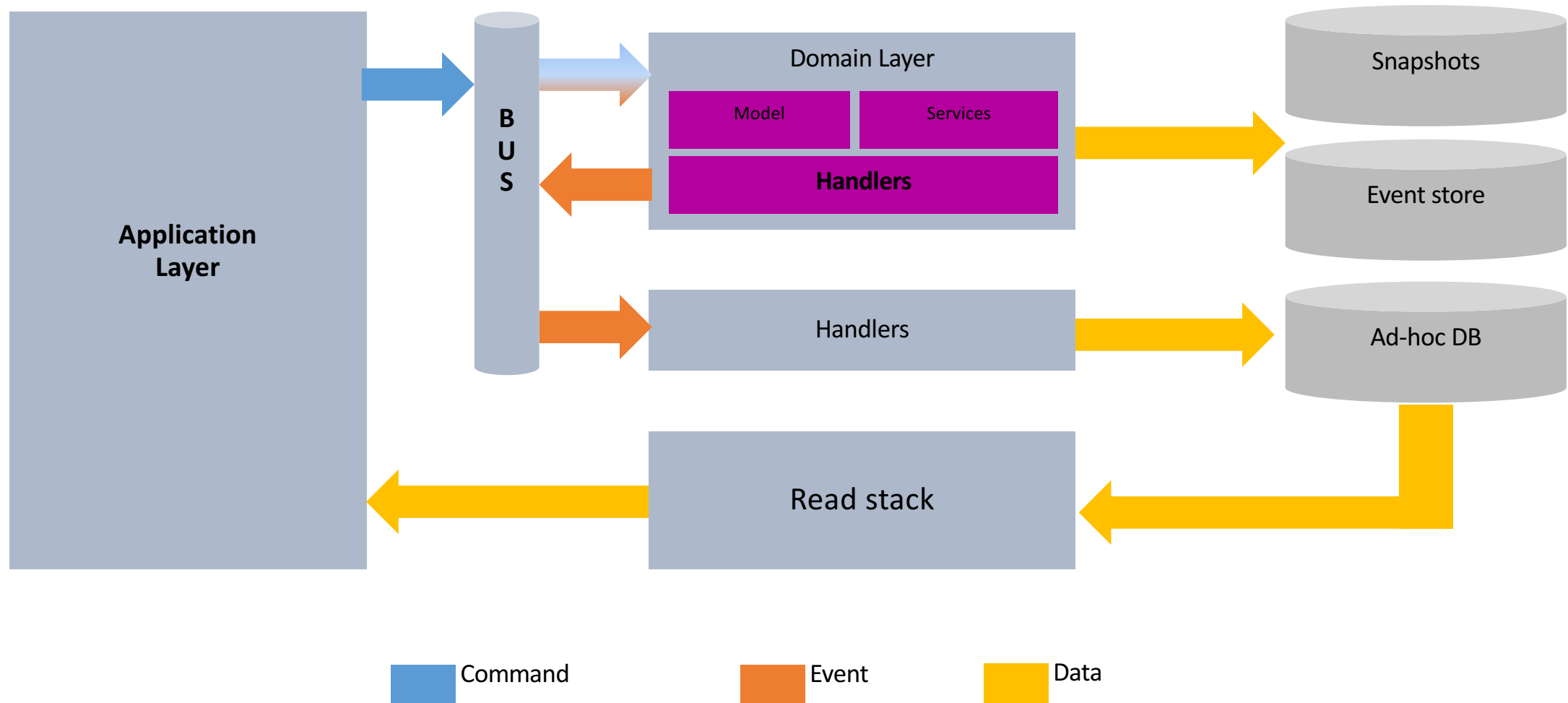
Read Model

# CQRS/ES in a nutshell

1. Application sends a command to the system
2. Command execution might alter the system's state and then raise events to state success/failure
3. Events are notified to interested subscribers (a.k.a. handlers), such as:
  - Workflow managers (a.k.a. «Sagas») which could execute more commands
  - Denormalizers, which will update the read model database

Note: command/event dispatch/execution will usually be managed by a Mediator («bus»)

# CQRS/ES at a glance

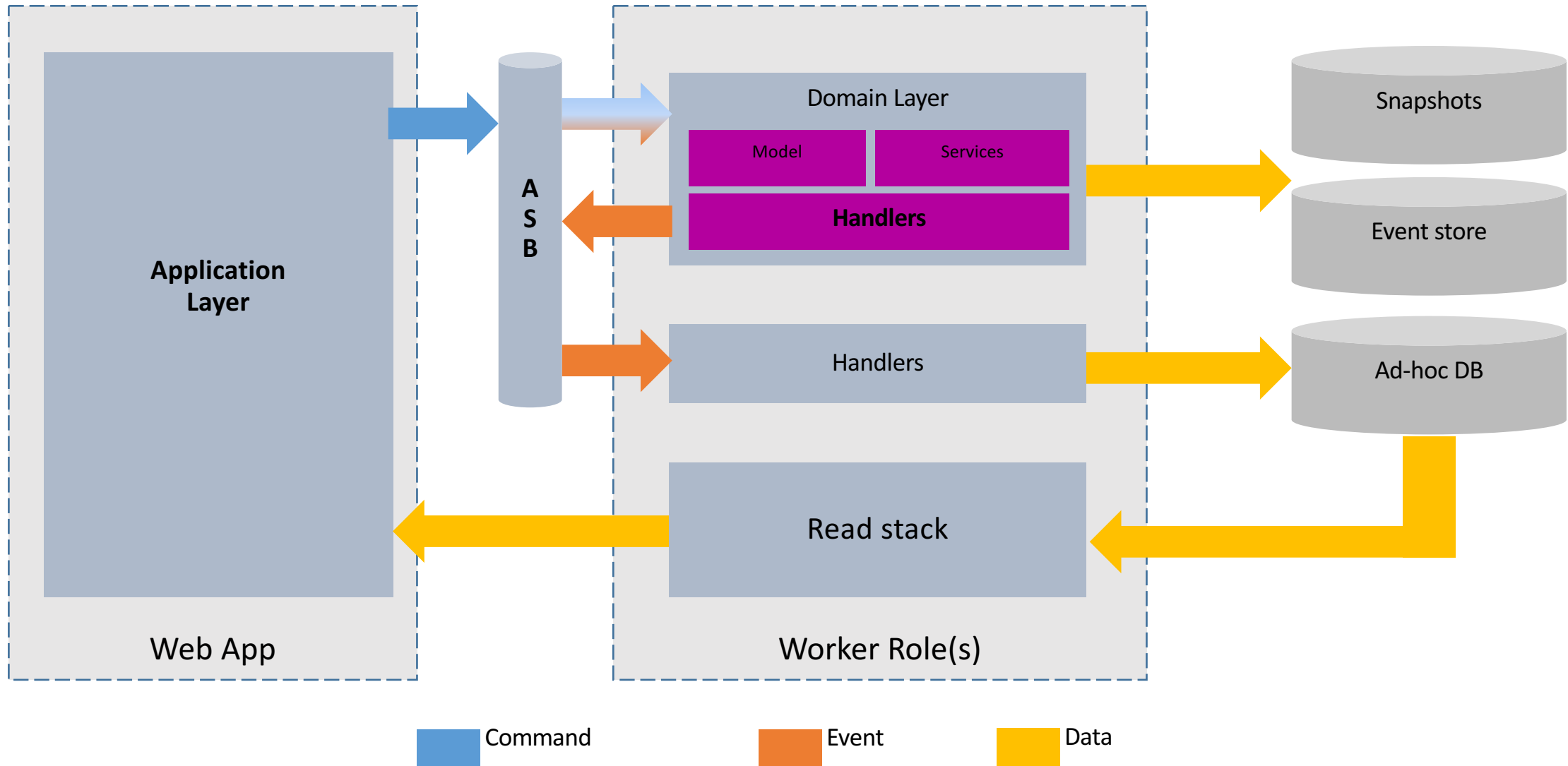


# DEMO

Handlers



# Azure <3 CQRS



# Look ma, a microservice!

Both web App(s) and worker role(s) can be:

- Evolved
- Deployed
- Configured (e.g.: scaled)

independently

# Application Frameworks

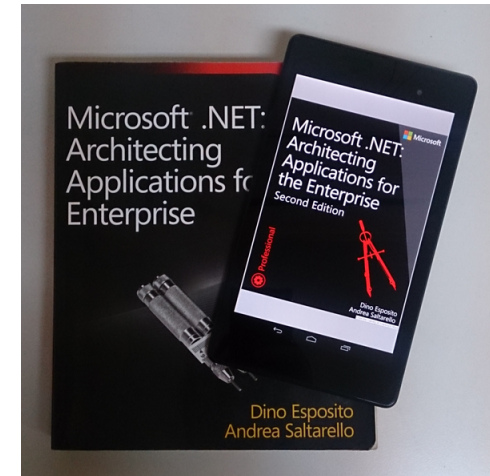
<b>MementoFX</b> <a href="https://www.nuget.org/packages?q=MementoFx">https://www.nuget.org/packages?q=MementoFx</a>	<b>NEventStore + CommonDomain</b> <a href="https://www.nuget.org/packages?q=NEventStore">https://www.nuget.org/packages?q=NEventStore</a>
<ul style="list-style-type: none"><li>• LGPL (FOSS in Q1-2017)</li><li>• DDD/CQRS/ES Full stack</li><li>• Time travel</li><li>• Alternative timelines</li></ul>	<ul style="list-style-type: none"><li>• FOSS</li><li>• DDD/CQRS/ES Full stack</li><li>• Both On-Premises &amp; cloud</li><li>• (Kind of) Wide community</li></ul>

# Bibliography

[DDD] Domain Driven Design, Eric Evans , Addison-Wesley

[NAAE] Microsoft .NET: Architecting Applications for the Enterprise (2° ed.), Andrea Saltarello & Dino Esposito, Microsoft Press

[MERP] Merp, <https://naa4e.codeplex.com/>



Domande?



